**Task -1:-** **Write a C program to reverse a given string without using any additional library functions.**

#include <stdio.h>

void revStr(char\* str)

{

int length = 0;

int i;

char temp;

for (length = 0; str[length] != '\0'; length++);

for (i = 0; i < length / 2; i++)

{

temp = str[i];

str[i] = str[length - i - 1];

str[length - i - 1] = temp;

}

}

int main()

{

char str[100];

printf("Enter a string: ");

scanf("%s", str);

revStr(str);

printf("Reversed string: %s\n", str);

return 0;

       }

**Task-3**:- **Explain the concept of structures in C and write a program to store student information (name, roll number, marks) using a structure.**

In C, a structure is a user-defined data type that allows the combination of data items of different kinds. Structures are used to group together different types of variables under a single name. Each variable within a structure is called a member.

The syntax to define a structure is:

struct structure\_name {

data\_type member1;

data\_type member2;

...

};

#include <stdio.h>

struct Student {

char name[50];

int rollNumber;

float marks;

};

int main() {

struct Student student;

printf("Enter student name: ");

scanf("%s", student.name);

printf("Enter roll number: ");

scanf("%d", &student.rollNumber);

printf("Enter marks: ");

scanf("%f", &student.marks);

printf("\nStudent Information:\n");

printf("Name: %s\n", student.name);

printf("Roll Number: %d\n", student.rollNumber);

printf("Marks: %.2f\n", student.marks);

    return 0;

}**Task-4:- Differentiate between single-linked lists and doubly-linked lists in C. Write code snippets to create a node and perform a basic insertion operation in a singly-linked list.**

Differences Between Singly-Linked Lists and Doubly-Linked Lists in C

|  |  |
| --- | --- |
| **Singly-linked Lists** | **Doubly-linked Lists** |
| Each node contains data and a pointer to the next node. | Each node contains data, a pointer to the next node, and a pointer to the previous node. |
| Navigation is only possible in one direction (from the head to the tail). | Navigation is possible in both directions (forward and backward). |
| Less memory usage compared to doubly-linked lists because each node has only one pointer. | More memory usage because each node has two pointers. |
| Insertion and deletion operations are simpler and slightly faster since only one pointer needs to be updated. | Insertion and deletion operations are more complex as both next and previous pointers need to be updated. |

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode) {

printf("Memory allocation error\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

**Task-5**:- **Explain the concept of pointers in C and write a program to swap the values of two variables using pointers.**

A pointer is a variable that stores the memory address of another variable. Pointers provide a powerful and flexible way to manipulate data and memory, allowing for dynamic memory allocation, efficient array handling, and the creation of complex data structures like linked lists and trees.

**Key Concepts:**

1. **Declaration**: A pointer is declared using the asterisk (\*) symbol. For example, int \*ptr; declares a pointer to an integer.
2. **Address-of Operator (&)**: The address-of operator is used to get the memory address of a variable. For example, ptr = &var; assigns the address of var to the pointer ptr.
3. **Dereference Operator (\*)**: The dereference operator is used to access the value stored at the memory address pointed to by the pointer. For example, \*ptr gives the value of the variable that ptr points to.
4. **Pointer Arithmetic**: Pointers can be incremented or decremented to point to the next or previous memory location.

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x, y;

printf("Enter value for x: ");

scanf("%d", &x);

printf("Enter value for y: ");

scanf("%d", &y);

printf("Before swapping: x = %d, y = %d\n", x, y);

swap(&x, &y);

printf("After swapping: x = %d, y = %d\n", x, y);return 0;

}

Pointer Arithmetic:

1. Write a C program to create an integer array of size 5, initialize it with values from 1 to 5, and then use pointer arithmetic to print each element of the array.

#include <stdio.h>

int main() {

int arr[5] = {1, 2, 3, 4, 5};

int \*ptr = arr;

for (int i = 0; i < 5; i++) {

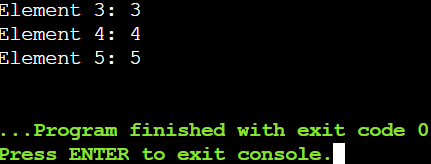
printf("Element %d: %d\n", i+1, \*(ptr + i));

}

return 0;

}

Output



Pointer to Pointer:

1. Write a C program to create a pointer to a pointer for an integer variable. Initialize the integer variable with a value, and then print its value using both the single pointer and the pointer to pointer.

#include <stdio.h>

int main() {

int num = 10;

int \*ptr = &num;

int \*\*ptr\_to\_ptr = &ptr;

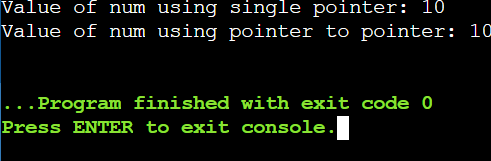
printf("Value of num using single pointer: %d\n", \*ptr);

printf("Value of num using pointer to pointer: %d\n", \*\*ptr\_to\_ptr);

return 0;

}

Output



Pointer Function Parameters:

1. Write a C function void swap(int \*a, int \*b) that swaps the values of two integers. Then, write a main function to test this swap function using pointer arguments.

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main() {

int x = 5, y = 10;

printf("Before swap:\n");

printf("x = %d\n", x);

printf("y = %d\n", y);

swap(&x, &y);

printf("\nAfter swap:\n");

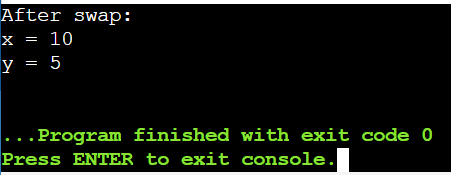
printf("x = %d\n", x);

printf("y = %d\n", y);

return 0;

}

Ouput



Dynamic Memory Allocation:

1. Write a C program to dynamically allocate memory for an array of integers of size 10. Initialize the array with values from 1 to 10, then print the values and free the allocated memory.

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*arr;

int size = 10;

int i;

arr = (int \*)malloc(size \* sizeof(int));

if (arr == NULL) {

printf("Memory allocation failed. Exiting...\n");

return 1;

}

for (i = 0; i < size; i++) {

arr[i] = i + 1;

}

printf("Array elements:\n");

for (i = 0; i < size; i++) {

printf("%d ", arr[i]);

}

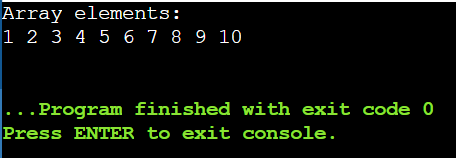
printf("\n");

free(arr);

return 0;

}

Output



Pointer to Function:

1. Write a C program to create a function pointer that points to a function int add(int, int). Use the function pointer to call the add function and print the result.

#include <stdio.h>

int add(int a, int b) {

return a + b;

}

int main() {

int (\*ptr\_add)(int, int);

ptr\_add = add;

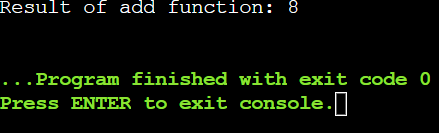
int result = ptr\_add(5, 3);

printf("Result of add function: %d\n", result);

return 0;

}

Ouput



Functions

Recursive Function:

1. Write a C function int factorial(int n) that calculates the factorial of a given number using recursion. Test this function in the main program by calculating and printing the factorial of 5.

#include <stdio.h>

int factorial(int n) {

if (n == 0) {

return 1; // Base case: factorial of 0 is 1

} else {

return n \* factorial(n - 1); // Recursive case

}

}

int main() {

int number = 5;

int result;

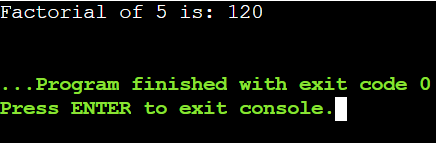
result = factorial(number);

printf("Factorial of %d is: %d\n", number, result);

return 0;

}

Ouput



Array of Function Pointers:

1. Write a C program to create an array of function pointers, where each function takes two integers as arguments and returns an integer. Include functions for addition, subtraction, multiplication, and division. Use the array to perform these operations on two integers and print the results.

#include <stdio.h>

int add(int a, int b) {

return a + b;

}

int subtract(int a, int b) {

return a - b;

}

int multiply(int a, int b) {

return a \* b;

}

int divide(int a, int b) {

if (b != 0) {

return a / b;

} else {

printf("Error: Division by zero\n");

return 0;

}

}

int main() {

int (\*operations[4])(int, int) = {add, subtract, multiply, divide};

int x = 10, y = 5;

int result;

for (int i = 0; i < 4; i++) {

result = operations[i](x, y);

switch (i) {

case 0:

printf("Addition: %d + %d = %d\n", x, y, result);

break;

case 1:

printf("Subtraction: %d - %d = %d\n", x, y, result);

break;

case 2:

printf("Multiplication: %d \* %d = %d\n", x, y, result);

break;

case 3:

printf("Division: %d / %d = %d\n", x, y, result);

break;

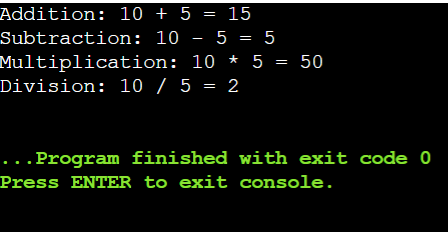
}

}

return 0;

}

Ouput



Function Returning Pointer:

1. Write a C function int\* createArray(int size) that dynamically allocates an array of integers of the given size and returns a pointer to the array. Initialize the array with values from 1 to size and print the array in the main function.

#include <stdio.h>

#include <stdlib.h>

int\* createArray(int size) {

int\* array = (int\*)malloc(size \* sizeof(int));

if (array == NULL) {

printf("Memory allocation failed\n");

exit(1); // Exit the program if memory allocation failed

}

for (int i = 0; i < size; i++) {

array[i] = i + 1;

}

return array;

}

int main() {

int size;

printf("Enter the size of the array: ");

scanf("%d", &size);

int\* array = createArray(size);

// Print the array

printf("The array is: ");

for (int i = 0; i < size; i++) {

printf("%d ", array[i]);

}

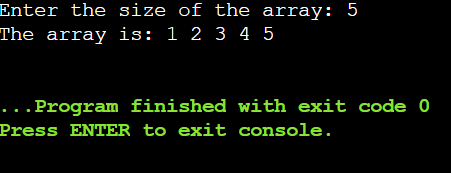
printf("\n");

free(array);

return 0;

}

Output



Higher-Order Functions:

1. Write a C function void applyFunction(int arr[], int size, void (\*func)(int \*)) that takes an array, its size, and a pointer to a function that operates on each element of the array. Write a sample function to double the value of each element and use applyFunction to apply it to an array.

Static Variables in Functions:

1. Write a C function that uses a static variable to count how many times the function has been called. Test this function in the main program by calling it multiple times and printing the count.

Structures

Structure Basics:

1. Define a structure struct Point with two integer members x and y. Write a C program to create a Point variable, initialize it with values, and print the values.

#include <stdio.h>

struct Point {

int x;

int y;

};

int main() {

struct Point p;

p.x = 10;

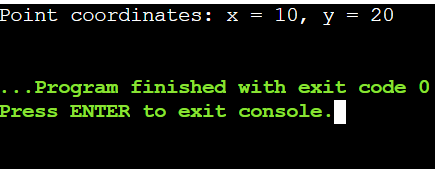
p.y = 20;

printf("Point coordinates: x = %d, y = %d\n", p.x, p.y);

return 0;

}

Ouput



Array of Structures:

1. Write a C program to define a structure struct Student with members name, age, and marks. Create an array of 3 students, initialize them with values, and print the details of each student.

#include <stdio.h>

struct Student {

char name[50];

int age;

float marks;

};

int main() {

struct Student student[3];

snprintf(student[0].name, sizeof(student[0].name), "Mahesh");

student[0].age = 24;

student[0].marks = 80;

snprintf(student[1].name, sizeof(student[1].name), "Prabhas");

student[1].age = 23;

student[1].marks = 91;

snprintf(student[2].name, sizeof(student[2].name), "Tarak");

student[2].age = 18;

student[2].marks = 100;

for (int i = 0; i < 3; i++) {

printf("Student %d:\n", i + 1);

printf("Name: %s\n", student[i].name);

printf("Age: %d\n", student[i].age);

printf("Marks: %.2f\n", student[i].marks);

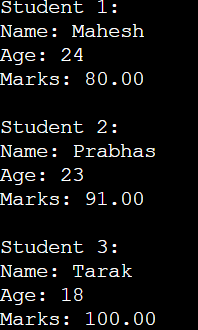
printf("\n");

}

return 0;

}

Ouput



Structure Pointers:

1. Write a C program to define a structure struct Rectangle with members length and width. Create a pointer to a Rectangle variable, dynamically allocate memory for it, initialize the members, and print the values.

Nested Structures:

1. Write a C program to define a structure struct Date with members day, month, and year, and another structure struct Student with members name and birthdate of type struct Date. Create a Student variable, initialize it with values, and print the student's details including the birthdate.

#include <stdio.h>

struct Date {

int day;

int month;

int year;

};

struct Student {

char name[50];

struct Date birthdate;

};

int main() {

struct Student student;

snprintf(student.name, sizeof(student.name), "Mahesh");

student.birthdate.day = 16;

student.birthdate.month = 12;

student.birthdate.year = 2000;

printf("Student Details:\n");

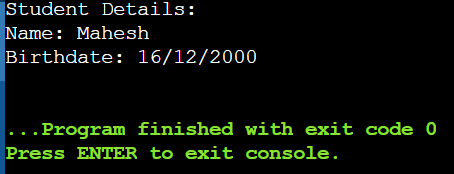
printf("Name: %s\n", student.name);

printf("Birthdate: %02d/%02d/%04d\n", student.birthdate.day, student.birthdate.month, student.birthdate.year);

return 0;

}

Output



Structure as Function Argument:

1. Write a C function void printStudent(struct Student s) that takes a Student structure as an argument and prints the details. Define a Student structure in the main program, initialize it with values, and call printStudent to print the student's details.

#include <stdio.h>

struct Student {

char name[50];

int age;

float marks;

};

void printStudent(struct Student s) {

printf("Student Details:\n");

printf("Name: %s\n", s.name);

printf("Age: %d\n", s.age);

printf("Marks: %.2f\n", s.marks);

}

int main() {

struct Student student;

snprintf(student.name, sizeof(student.name), "Mahesh");

student.age = 24;

student.marks = 88;

printStudent(student);

return 0;

}

Output

